MEASURING THE IMPACT OF COOPERATIVE REWARDS ON AI

By

Dain Harmon, B.S.

A Project Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science

University of Alaska Fairbanks

December 2020

APPROVED:

Orion S Lawlor, Committee Chair

Glenn G Chappell, Committee Member

Jonathan B Metzgar, Committee Member

Chris M Hartman, Chair

Department of Computer Science

Abstract

We consider the effects of varying individualistic and team rewards on learning for a Deep Q-Network AI in a multi-agent system, using a synthetic team game 'Futlol' designed for this purpose. Experimental results with this game using the OpenSpiel framework indicate that mixed reward structures result in lower win rates. It is unclear if this is due to faster learning on simpler reward structures or a flaw in the nature of the reward system.

Table of contents

1) Introduction

The use of AI agents is increasing world wide. From shopping recommendations to self-driving cars and medical diagnoses AI agents are showing up in almost every industry. It is thus important that these agents “play nice” with each other in a way that is beneficial to humans. Even single agent systems can produce poor results considering the outrage promoting behavior of the Facebook feed and YouTube recommendation algorithms (Munn 2020).

We must thus be even more concerned about the result of multi-agent interactions as the multiple agents make the system more complicated and more unpredictable than the already difficult to predict behavior of existing AI.

We here take a look at the effects of changing the incentives an AI is operating under on its ability to learn a task involving multiple agents. We focus on the effects of individual rewards as compared to collectivized rewards for a team of agents playing a game of our own making for this purpose.

2) Background

Perceptron

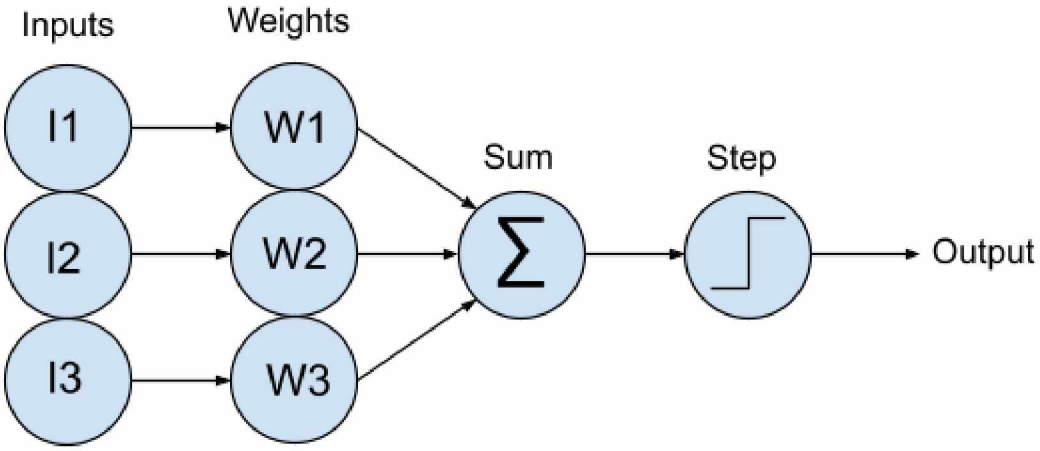


Figure 2.1 Perceptron

Perceptrons (Rosenblatt, 1957) are one of the simplest forms of neural network with a single layer and a single artificial neuron. Perceptrons are an algorithm for learning binary classifiers as their final output is either a 1 or a 0 indicating membership in a class.

Perceptrons work by taking in an input vector **I** that represents the data to be classified. This vector is then multiplied by a weight vector **W** and then summed. This sum then has an activation function applied to it. This activation function forces the output to 0 or 1 depending on the weighted sum of the inputs (and also possible to add a bias to make triggering the activation function easier). They can be thought of mathematically thusly:

$$f(x)=\begin{cases}1 & b+\sum_0^m x_i w_i>0\\ 0 & otherwise\end{cases} \qquad 2.1$$

where *x* is the input vector, *b* is the bias, *w* is the weight vector and *m* is the size of the input vector.

Perceptrons are only able to learn effective classifications for things that can be linearly separated (Minsky and Papert 1972). That is the classes can be separated by drawing a straight line through the data points. Multiple nodes can be combined using various mechanisms to create more complex patterns. Other activation functions such as sigmoids might be used for situations where continuous output is preferred. However, it is by making layers of perceptrons that one truly escapes the limits of the basic Perceptron.

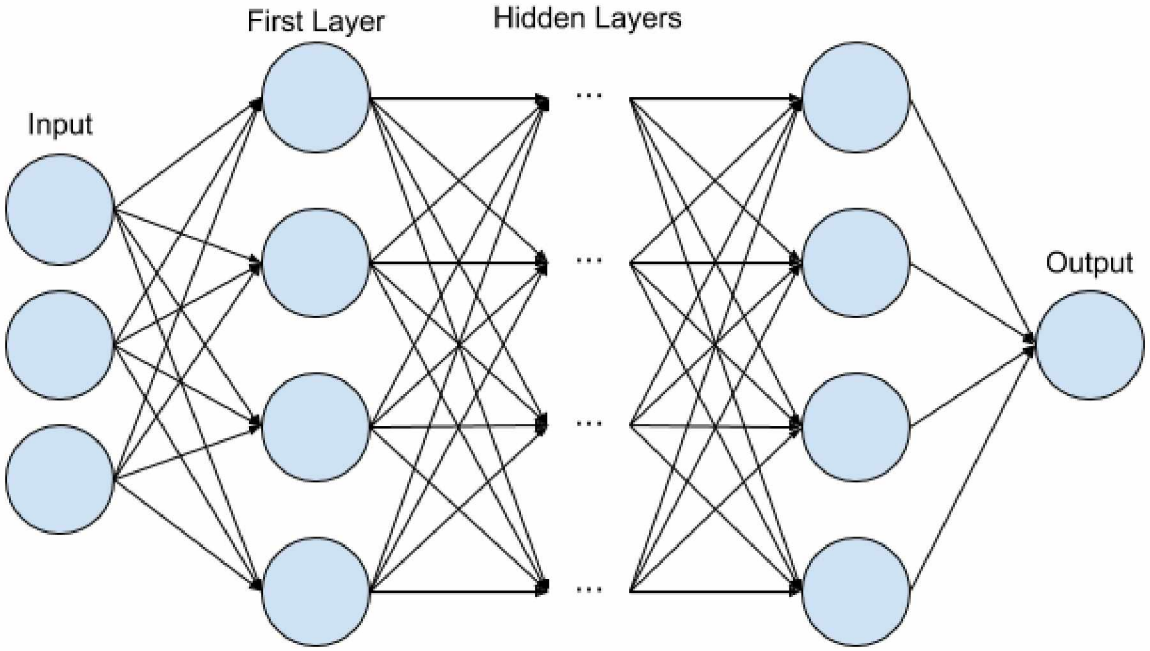


Figure 2.2 Feedforward Neural Network

A neural network (Ivakhnenko and Lapa, 1965) is made up of layers of artificial neurons. In the simple feedforward neural network, the output of the previous layer is then used as the input for the next layer. The final layer's output is then used to complete whatever task the neural network was meant for. Layers may have varying levels of connectivity from total to very little.

Other types of networks exist with connections bypassing layers or even backward in recurrent neural networks such that the neuron's output is included in future activations creating a form of internal memory useful for text and other linear sequence tasks. Convolutional neural networks iterate a convolutional kernel over the input to extract local features making them effective image classifiers.

Deep neural networks are named such due to the use of multiple layers to extract increasingly high-level features from the input. Low layers might find edges of an object while higher layers use the arrangement of edges to conclude that a person is or isn't present. Convolutional neural networks are usually arranged this way.

Neural networks operate as powerful general function approximators and can allow extremely complex functions to be compressed down to a simple mass of linear algebra. Furthermore using evolutionary algorithms or other learning techniques like backpropagation a completely unknown function may be learned by a neural network.

Q-Learning

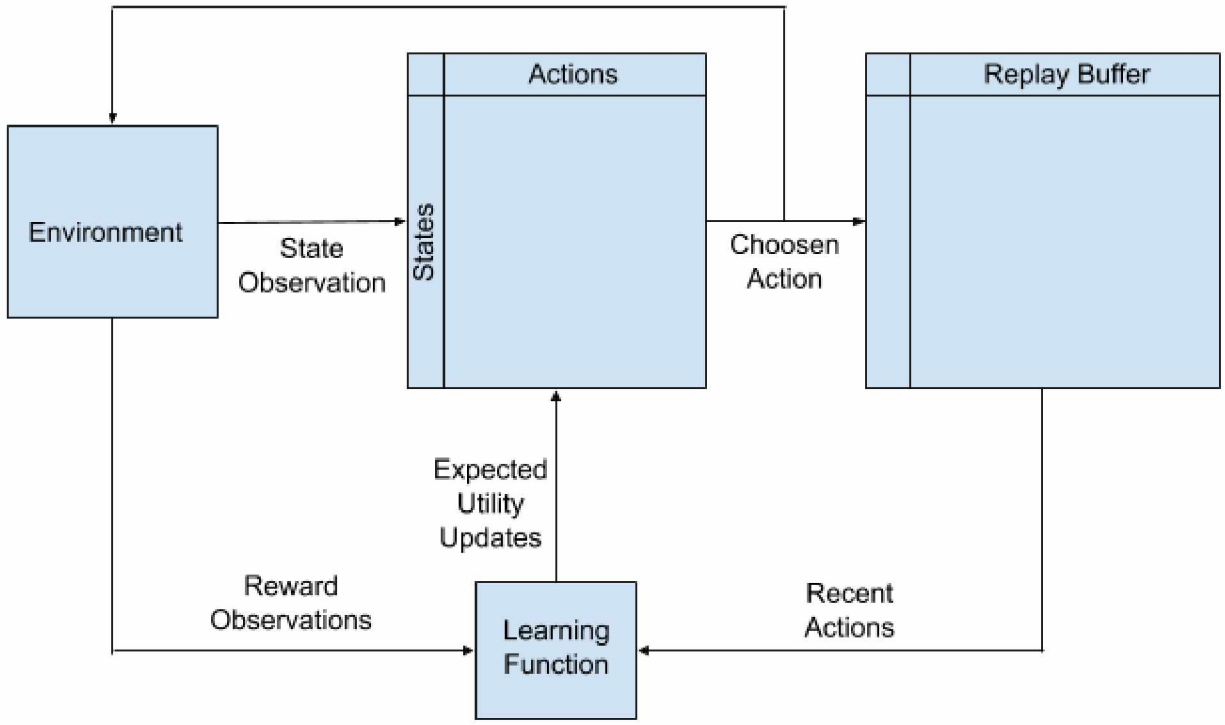


Figure 2.3 Q-Learning

Q-Learning (Watkins and Dayan, 1992) is a table-based reinforcement learning algorithm. A table is initialized to zero at the beginning of learning. The table contains entries for every combination of world state and action. Additionally, a record of past action-state combinations is also recorded in a replay buffer. At each step, the table is consulted to determine the highest reward action for the currently observed state with some random factor included encouraging exploration.

Early on the agent is acting more or less randomly. However, as the agent interacts with the environment it will get feedback in the form of utility rewards. These rewards can be used to update the table with new values, typically by adding the latest reward value damped by a learning rate to the currently stored value. Using the replay buffer, previous steps leading up to one that gained a reward can also be updated with temporally damped values allowing the agent to learn complex chains of actions leading up to a given reward.

DQN

The deep q-network combines a neural network with q-learning. The table in q-learning can quickly bloat out of control consuming an unreasonable or even impossible amount of space to store. The table also lacks any form of generalizability as each state-action pair must be learned

about individually. To counteract this in a deep q-network the table is replaced with a neural network. This neural network learns to approximate the idea table over the course of the learning process. A convolutional neural network is often used to exploit the generalizability of that type of neural network. This technique was used by Google scientist to play Atari 2600 games at levels equivalent to expert humans (Mnih et al, 2015)

Cooperation and Conflict

Cooperation and conflict in AI have been explored before. OpenAI (OpenAI et al, 2019) got expert human level play from reinforcement learning AI in the Multiplayer Online Battle Arena (MOBA) game Dota2. In Dota2 5 players per team each choose a unique hero to control and engage the other team with the ultimate goal of destroying the enemy's home base. High-level Dota2 combat requires highly coordinated action. Human teams train to have an instinctive understanding of their teammate's actions and use several modes of communication to coordinate. OpenAI's AI the OpenAI Five does not communicate with its AI teammates. Instead, their expert cooperation is born of them all being copies of each other and thus knowing exactly what their teammates will do because it's what they would do. This does however require the AI need to have mastered all of the hero it chooses to play, one of the reasons the AI played games only with a reduced pool of heroes.

Deepmind (Leibo et al, 2017) explored cooperation and conflict in the context of social dilemma games ala the prisoner's dilemma. They created two games. In the first, two agents compete to collect "apples" in a 2D gridworld. Agents were given the ability to tag the other agent and remove them from the grid temporarily, preventing them from taking any apples. In the other game, two "wolf" agents hunt a third "pray" agent. The wolf agents were able to share a reduced credit for the kill by being near the kill when it was made or take full credit for themselves if the other wolf was not near. It was found that more complex "smarter" agents were more likely to be aggressive in the first game, but more likely to share credit in the second.

Deepmind (Anthony et al, 2020) also continued earlier work (Paquette et al, 2019) on testing AI on the board game Diplomacy.[1] Diplomacy is specifically designed to emphasize the tension between cooperation and conflict and has a reputation for ruining friendships due to how finely balanced the game is. Diplomacy uses simultaneous turns to create the opportunity for surprise betrayal. Each player records their order secretly after a period of discussion with the other players and then these orders are revealed at the same time and executed. Cooperation is key to early success but there are many opportunities throughout the game to betray a former ally for strategic advantage. Deepmind had success in developing an improved AI for playing a reduced version of Diplomacy without communication and though they had provided a good baseline for further work using the full version of Diplomacy to explore communication and alliance making in AI.

In the 3D first person shooter game *Quake III Arena,* Jaderberg et al (2019) succeeded in getting a group of agents to play the game's multiplayer Capture the Flag mode at a superhuman level. The AI learned to work with its teammates to guard the home base/flag, opportunistically wait for the target flag to reappear and to follow the flag carrying player.

---

[1] We considered using Diplomacy as a game to explore cooperation ourselves before Deepmind's paper came out but rejected it due to its complexity.

3) Experiment

Objective

The core goal of this experiment was to determine the effects of teams of agents using different reward/utility structures per team on a global group success metric. In a game environment the highest metric is win rate, whether this a simple win rate, win/loss ratio or the more complicated MMR used in chess and many competitive online games. It is hard to argue with victory. Thus we use the win rate as our global metric. For this experiment we considered the effects of individualistic rewards vs group rewards on the win rate.

OpenSpiel

OpenSpiel is a framework for reinforcement learning in games created by Google DeepMind (Lanctot et al, 2019). It contains a collection of algorithms, environments and games meant for research in reinforcement learning and search/planning in games. OpenSpiel supports a variety of game types. The framework supports games that have an arbitrary number of players. Games may be zero-sum, cooperative or general-sum. One-shot games are possible along with sequential games, both with individual turns or or with multiple turns occurring simultaneously. Games may also have limited information or random elements. OpenSpiel currently has over 45 implemented games including a variety of different combinations of the above options

OpenSpiel uses C++ for the game implementations. The games are then wrapped in Python allowing algorithms to be implemented in C++ or Python using a nearly identical API. Tensorflow is used to support most of the algorithms written in Python. Tensorflow provides CUDA (version 10.1 was used) GPU support which can considerably accelerate the learning and evaluation process of neural-networks due to the highly parallel nature of the GPU.

Over 25 algorithms exist in the current implementation of OpenSpiel ranging in type from simple Searches to Reinforcement Learning algorithms. Of particular interest is the Deep Q-Network which combines the power of a neural-network with the utility/reward structure of Q-Learning.

Game/Fotlol

To explore the effects of individualistic rewards vs group rewards we needed a game that can support that. We needed a game with multiple players where cooperative rewards were possible. We were inspired to develop our game by games like capture the flag or american football which provide space for both group action and individual glory.

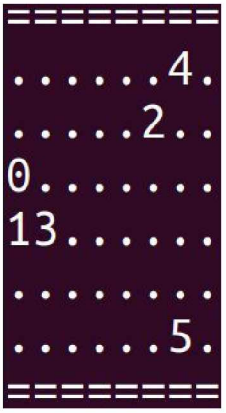


Figure 3.1 A Futlol Game Board

Futlol[2] is the game we developed. Futlol is plated on an 8x8 game board. Each team has a goal zone on their side of the board indicated by '=' symbols in the image to the right. Teams are split into even numbered players playing as a team with their goal at the top and odd players playing as a team with their goal in the bottom. A '.' indicates empty spaces. For our experiment we had teams of 3 Players each though that number could easily be increased.

Players take turns in numeric order Player 0 -> Player 1 -> Player 2 -> Player 3 -> Player 4 -> Player 5 -> Player 0 -> …, with Player 0 always going first.

At the beginning of the game no players are on the board. A player off the board may choose to enter the board at any unoccupied space in their team's goal zone. Once on the board a player may use their turn to move to any valid unoccupied space adjacent or diagonal to their current space or they may choose to stay still.

A player may also move to 'tackle' an enemy piece as long as the tackler starts their turn on their half of the board (top 4 rows for even, bottom 4 rows for odd). A tackled piece is removed from the board and must reenter the board just like at the beginning of the game. This tackling option allows for defensive action other than bodyblocking and also creates increased risk for attackers. It also creates a situation which may be rewarded or pushed. A clever player may also note that getting captured allows one to teleport back to their goal which could be used defensively.

Goals are scored by moving into the enemy's goal zone. For our experiment 3 goals where required to win a game. Players can also time out the game after taking 2400 moves each.

This construction allows for both offensive and defensive strategies and also allows for rewardable events. A player may be rewarded or punished for movement, capturing, being captured, or scoring a goal. These events can also be considered from both a team and individual perspective.

Agents

For our experiment each player was controlled by a DQN agent using the DQN algorithm provided in the OpenSpiel framework. Each agent had two hidden layers each with 64 nodes each. Agents also had a replay buffer of 3,333 previous moves and the resulting rewards

[2] Originally proposed and named by Dr. Orion Lawlor

(Increased to 10,000 after round 3640). For each learning step 32 previous steps were sampled. Each agent was provided with a full view of the current board and all of the valid moves from their current position.

Reward Structure

OpenSpiel allows for reward values to be updated every turn. We used the values in Table 3.1 to reward agents.

Table 3.1 Reward Structure

Action/Event	Utility Reward	Individualistic Player	Team Player	Win-Only Player
Move	0			
Capture	1	Self Only	Any Team Member	
Get Captured	-1	Self Only	Any Team Member	
Score Goal	50	Self Only	Any Team Member	
Win (3 Goals)	200		Scored	Scored

Capturing was set up to be a zero-sum game preventing teams from creating “gentlemen's agreements” where they trade captures to inflate their utility rewards. A previous training version used in testing allowing positive-sum capture ended up with widespread capture trading. This doesn’t prevent a superior team or a team close to winning from extending the game to increase the utility gain however.

Different agents consider the rewards differently. A player might be marked as a Win-Only player and only gain utility from winning the game. Other players use two values, a personal action/event and a team action/event percentage value, that determines how much they care about that type of reward. These percentage values are multiplied by the utility reward when the event occurs and then the utility is awarded to the player. Winning is considered a team reward.

An agent may thus be a purely individualistic player (100% personal reward, 0% team reward). A 2:1 individualistic player (100% personal reward, 50% team reward) or any other combination.

Teams

We created six teams(Team 0-5) in total covering a range of reward ratios and a win only team for comparison. Each agent has the same reward structure as it’s teammates. Teams are detailed in Table 3.1 Team Summary.

Table 3.2 Team Summary

Team	Personal Action/Event Reward	Team Action/Event Reward	Ratio	Win-Only
0	100%	0%	1:0	
1	100%	50%	2:1	
2	100%	100%	1:1	
3	50%	100%	1:2	
4	0%	100%	0:1	
5				Yes

Tournament

To train and evaluate the teams were placed into a continuous round robin tournament.

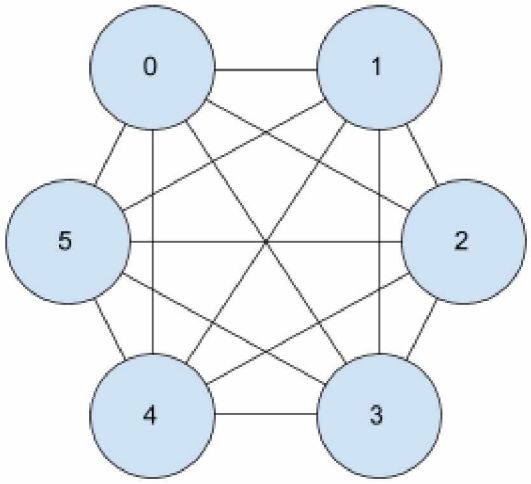


Figure 3.2 Round Robin Tournament Structure

Every round of the tournament each team faced each other once and played themselves one. Thus 21 games were played per round with each team playing 6 games. Evey round was considered a training round and also had evaluation statistics taken. Agent networks were backed up every 10 rounds to allow for resuming in the event of a crash or other fatal error. Also each backup round full game playthroughs were saved for review.

As the experiment was set up as a self competitive system and the objective based on comparison there were no separate evaluation rounds as we lacked a good independent evaluation. This did provide some benefit as we could get maximum efficiency out of each game played.

Hardware/Software

We ran the experiment on a PC with a Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz with 4 cores, 8GB of DDR3 RAM (Upgraded to 24GB after round 3640) and a NVIDIA GeForce GTX 1660 GPU (Driver Version: 455.23.05) all mounted on a ASRock Z97 Pro3 motherboard.

The PC ran Ubuntu Linux 20.04.1 LTS. We used OpenSpiel v0.2.0 and Tensorflow v2.2.1 with CUDA v10.1.

Data Collected

At the end of each game the following information was recorded:

- Time Stamp
- Round Number
- Teams Playing
- Ending Scores
- Captures per Player
- Times Captured per Player
- Goals Per Player
- Cells Moved per Player

In fully recorded games (occuring every 10^{th} round) the same was recorded along with board state for every turn taken.

After collection the data was processed to provide the following round by round information for each team:

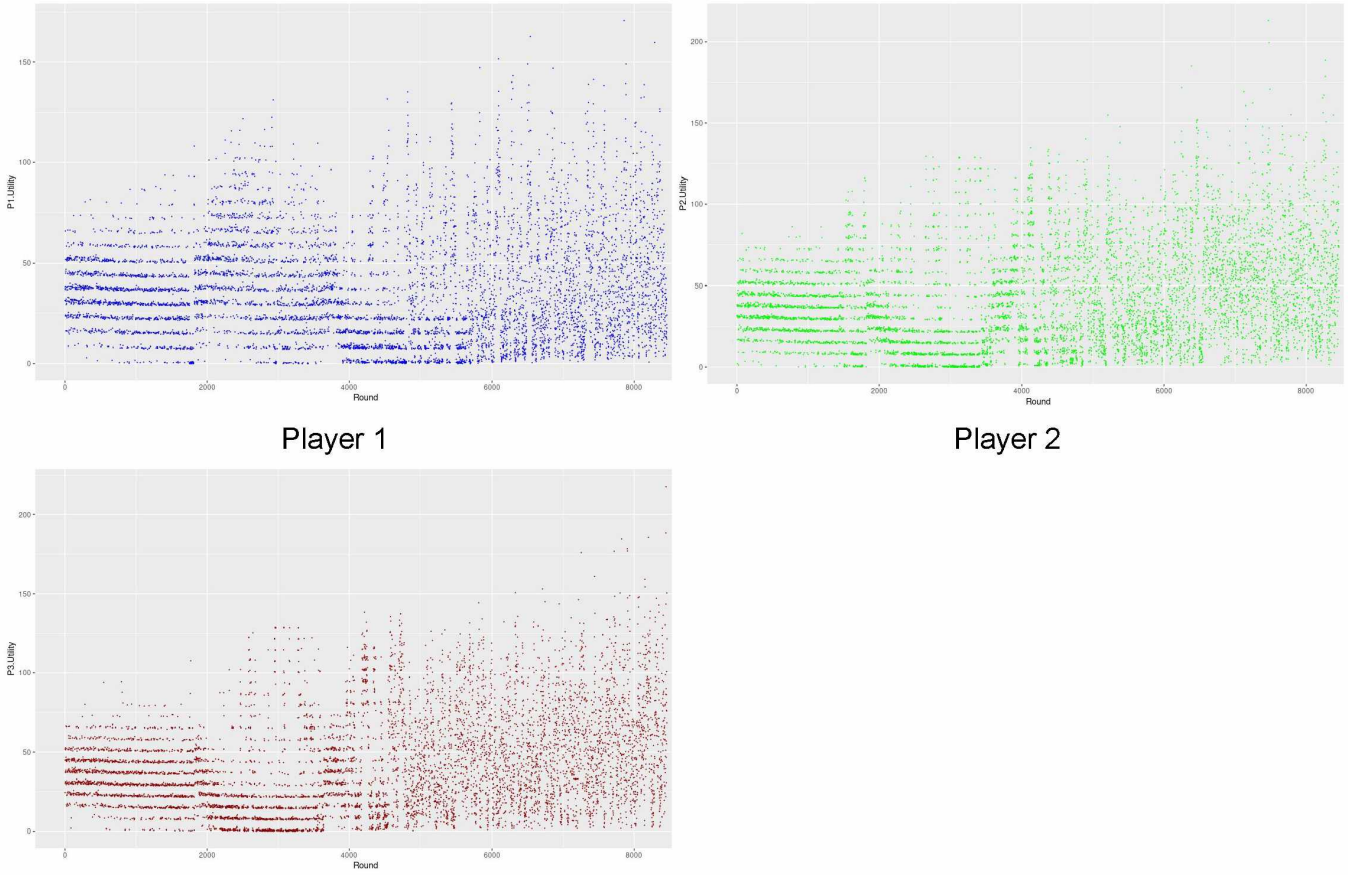
- Round
- Wins
- Losses
- Average goals scored per game for round
- Average captures per game for round
- Average times captured per game for round
- Average turns scored per game for round
- Utility scored per game for round
    - Maximum per game
    - Total of all players
    - Average of all players
    - Range per game
    - Total per player

4) Results

Due to the large amount of data collected, presented here are selected parts of the data relating to the core question we wished to consider.

Team 0 Player Utility vs Round

Figure 4.1 Team 0 Player Utility vs Round



Player 1

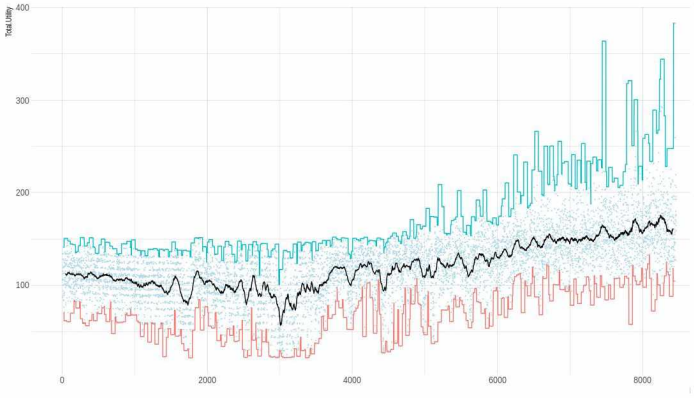
Player 2

Player 3

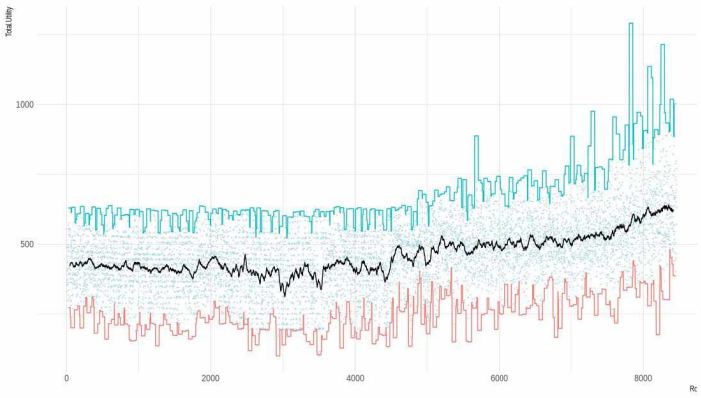
The players of Team 0 are primarily gaining utility from the 50 utility reward for scoring a goal early in the training cycle. The discrete jumps begin to fade into more continuous values at approximately round 4500 for Player 3. This happens later for Player 2 and Player 1 at approximately round 5000 and round 5750 respectively. This transition occurs due to the agents getting a lot more of the 1/-1 reward for captures blurring the discrete jumps from goals. This increase in captures at this time in training can be seen in further graphs. The agents while developing similar behaviors are learning them at different rates.

Team Average Total Utility vs Round

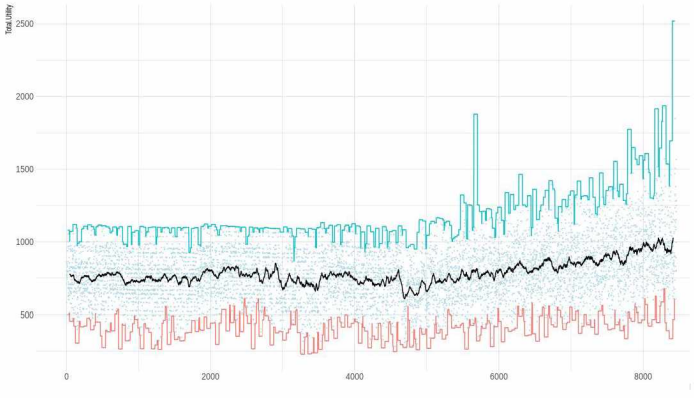
Figure 4.2 Team Average Total Utility vs Round



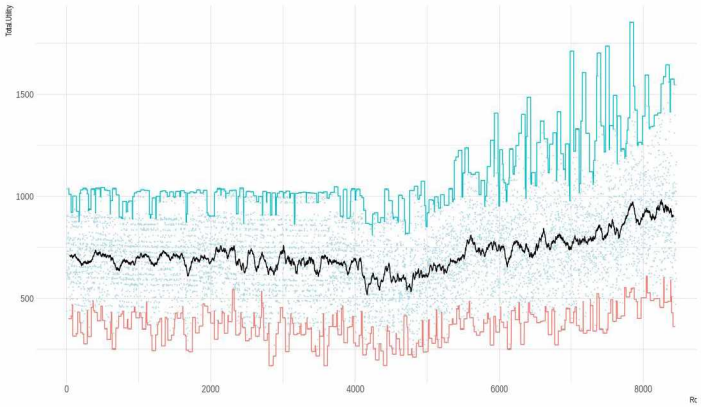
Team 0: Individualistic



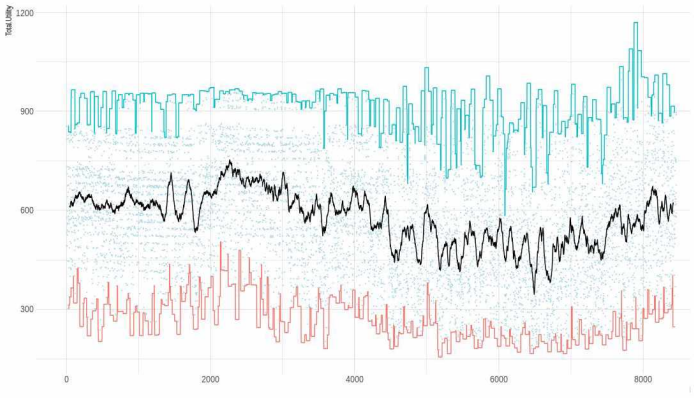
Team 1: 2:1 Individualistic



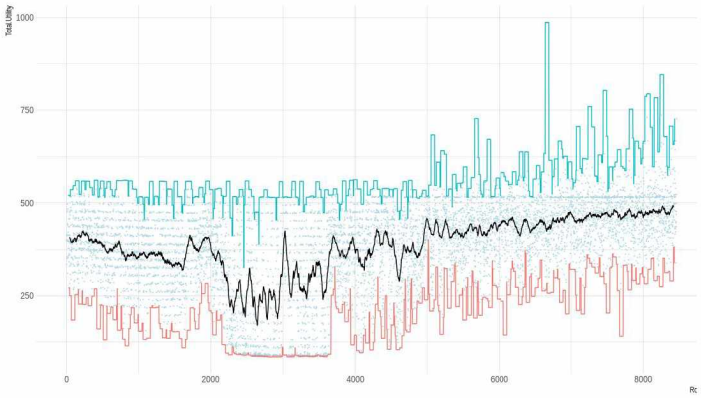
Team 2: 1:1 Individualistic



Team 3: 1:2 Individualistic



Team 4: Team Only



Team 5: Win Only

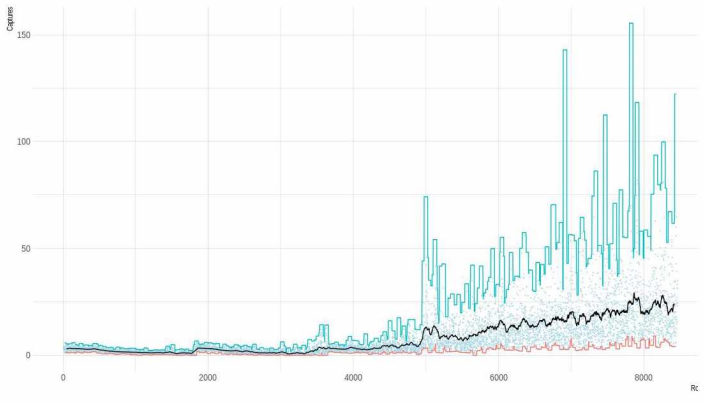
Due to the high level of noise in the data (light blue dots) it seems best to consider the minimum, maximum and average. In these graphs the blue line represents a local 50 point moving maximum and the red an equivalent minimum. The black line is a 90 point average.

Graphed is the sum of all three players on a team's utility which is then averaged over the 6 games in a round. All teams show an upward trend in the total utility starting around round 5000, save for Team 4, which seems to develop an upward trend in minimum and average at round 7000. All agent teams are thus successfully optimizing their utility functions. However, this does not imply that they are winning as there is an "endless" source of utility in captures for all teams save Team 5.

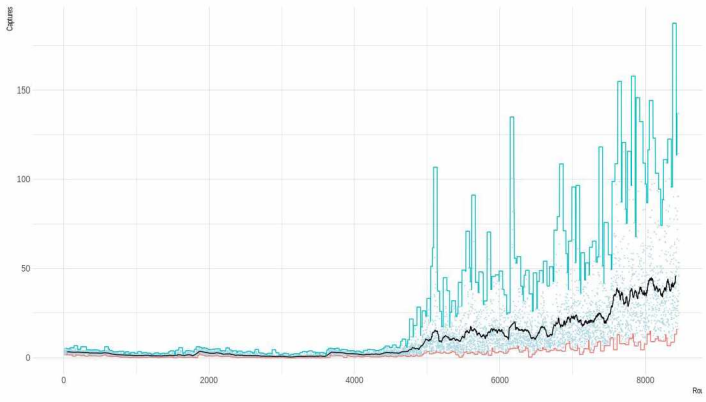
Also of note is the period where Team 5's minimum was at or near 0. Due to the tournament requiring each team to face itself for a team to score 0 it must timeout the game (and also avoid scoring and capturing for teams 0-4).

Average Team Captures Per Game vs Round

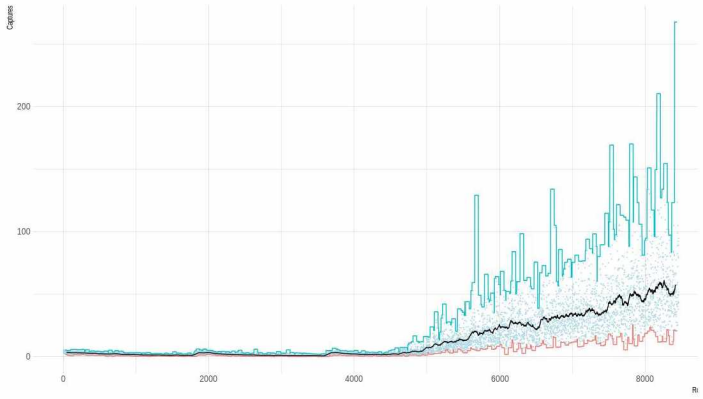
Figure 4.3 Average Team Captures Per Game vs Round



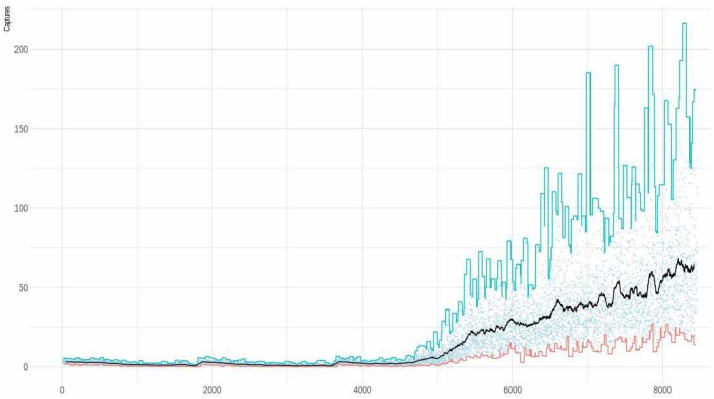
Team 0: Fully Individualistic



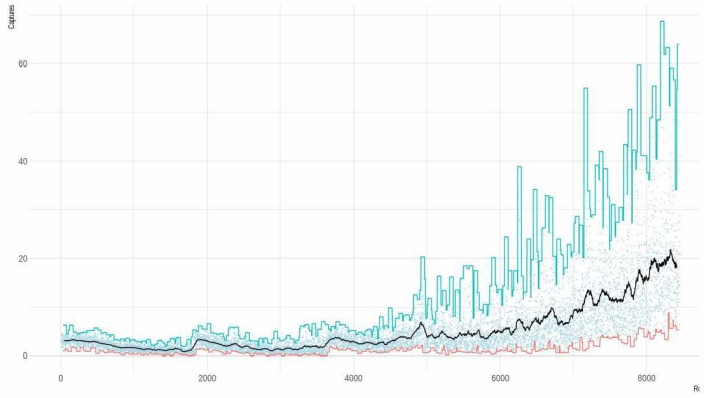
Team 1: 2:1 Individualistic



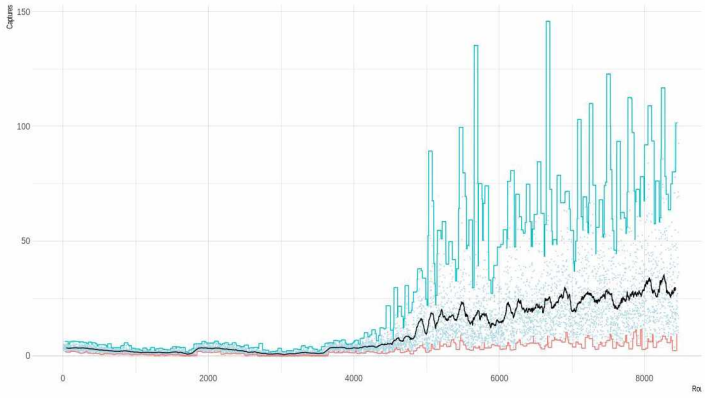
Team 2: 1:1 Individualistic



Team 3: 1:2 Individualistic



Team 4: Team Only

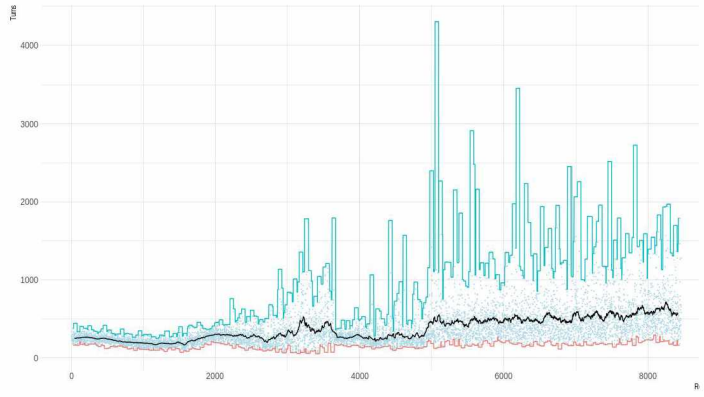


Team 5: Win Only

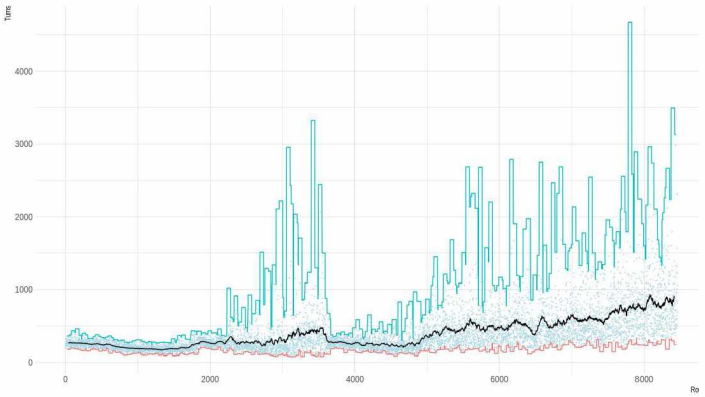
All teams show an increase in the number of captures made. Interestingly teams 1-3 have an ending average or around 50 captures per game; this is double the ending averages of the most successful teams in terms of win rate (Team 1 and Team 5 at around 25). The least successful team (Team 4) never has an average much beyond 20. This demonstrates that while captures are useful there is a limit to how much they help winning vs helping inflate a teams utility value.

Moves Per Game vs Round

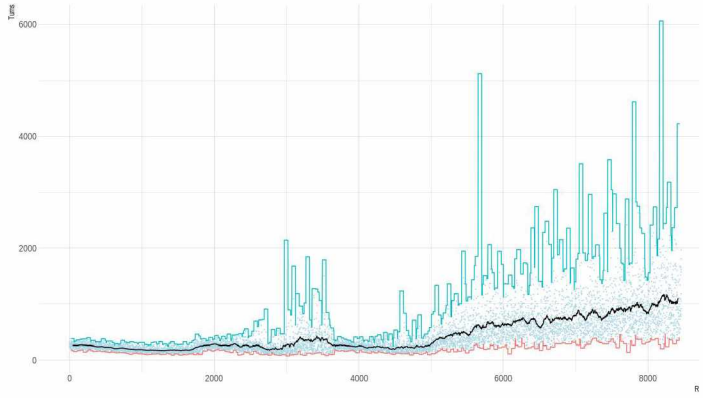
Figure 4.4 Moves Per Game vs Round



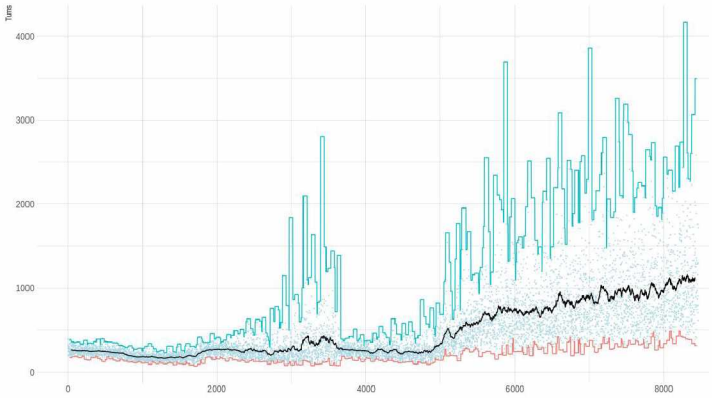
Team 0: Individualistic



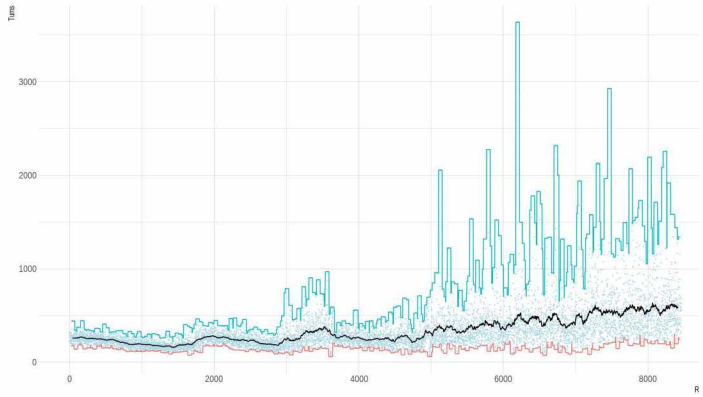
Team 1: 2:1 Individualistic



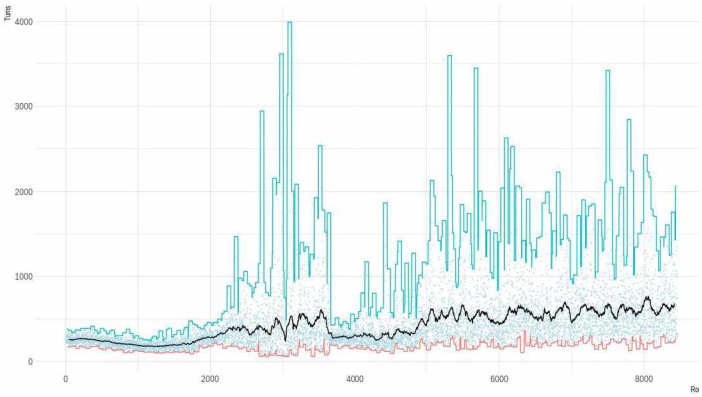
Team 2: 1:1 Individualistic



Team 3: 1:2 Individualistic



Team 4: Team Only

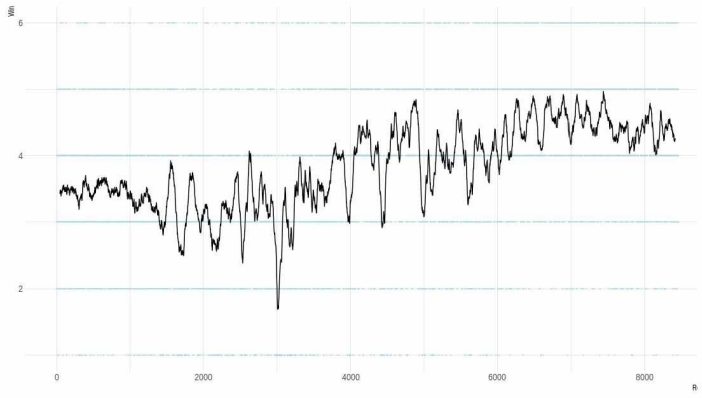


Team 5: Win Only

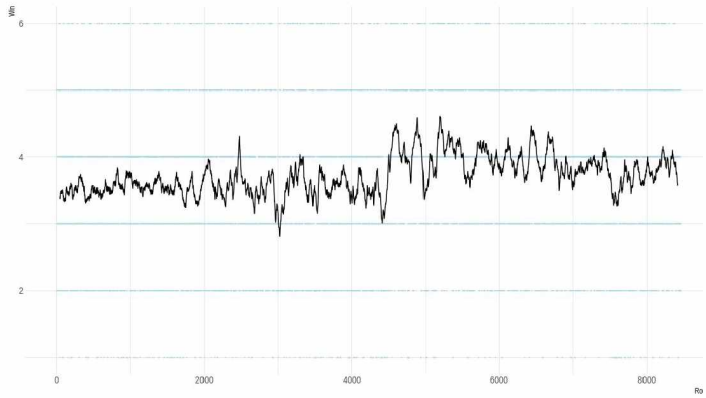
All teams also show an increase in game length as they learn more complex play. Also of interest is the falloff at round 3640. Training was resumed here from a saved network. However, the ~5gb FIFO replay buffer that stored past moves was not saved showing the impact of such a buffer on decision making and learning.

Wins Per Round

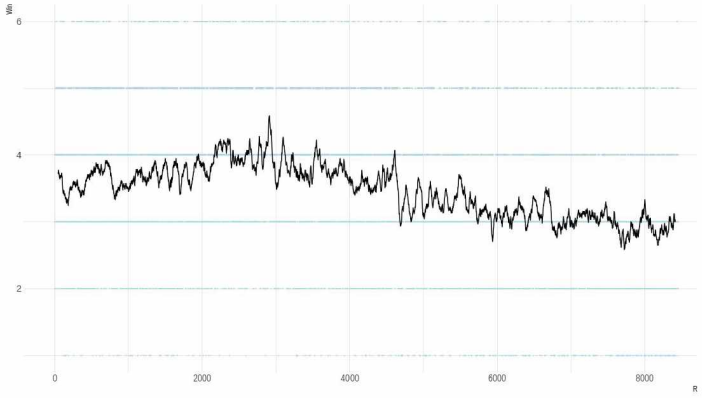
Figure 4.5 Wins Per Round vs Round



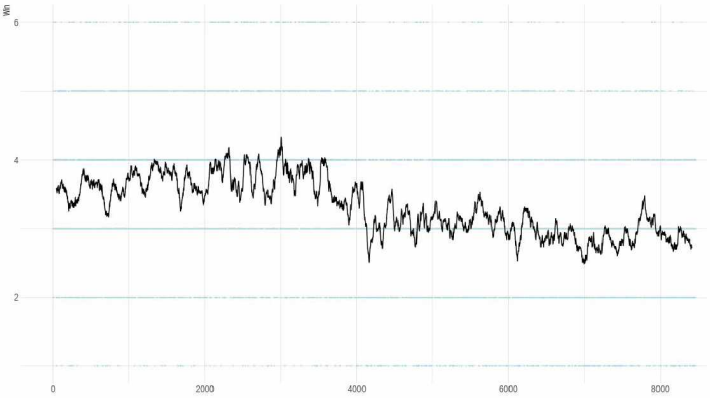
Team 0: Individualistic



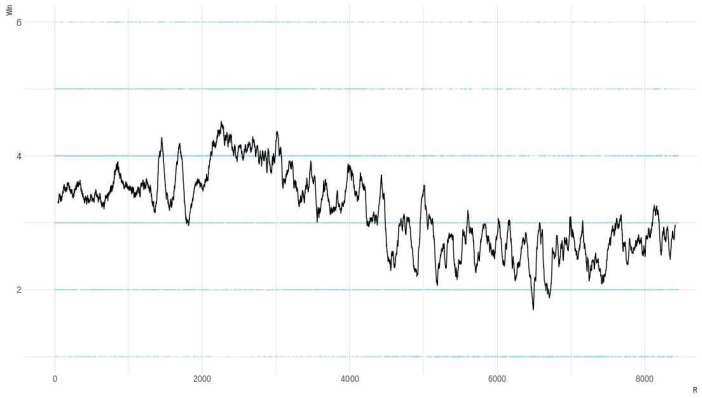
Team 1: 2:1 Individualistic



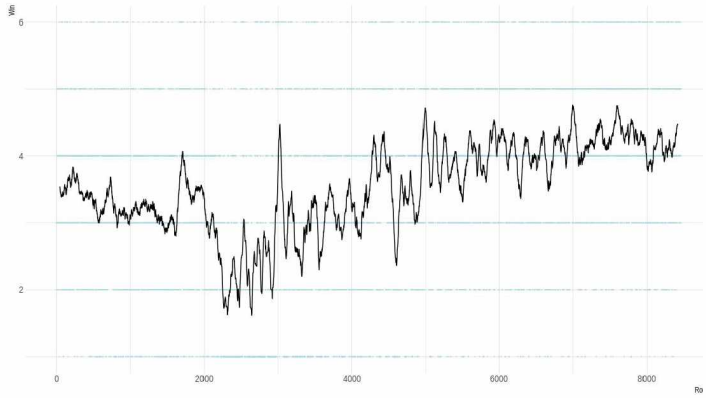
Team 2: 1:1 Individualistic



Team 3: 1:2 Individualistic



Team 4: Team Only



Team 5: Win Only

The collective success of the teams of agents is best shown by their win rate. Teams 0 and 5 are shown to be dominant for a period of over 1000 rounds. Interestingly Team 0 has no reward mechanics in common with Team 5.

Table 4.1 Average Wins per Round

Team	Average Wins per Round (Last 400 Rounds)
0	4.404
1	3.836
2	2.956
3	2.830
4	2.832
5	4.142

An average of the last 400 rounds shows the final state of the agent teams. Team 0 wins most often, followed by Team 5. Team 1 holds a convincing third place. Team 4 Just barely beats out Team 3 in avoiding last place.

Team Win Rate Vs. Team

Table 4.2 Team Win Rate Vs. Team (Last 400 Rounds)

		Winner					
		0	1	2	3	4	5
Loser	0		0.2375	0.25	0.32	0.32	0.505
	1	0.7625		0.2325	0.3775	0.3475	0.46
	2	0.75	0.7675		0.48	0.4375	0.655
	3	0.68	0.6225	0.52		0.5875	0.7875
	4	0.68	0.6525	0.5625	0.4125		0.805
	5	0.495	0.54	0.345	0.2125	0.195	

While this mostly follows the trends shown above there are some interesting details in the specifics of the team vs team win rate. Here we can see that Team 5 picks up most of more of its wins from Teams 3 and 4. Team 0 on the other hand gets more wins from games against Teams 1 and 2.

5) Conclusions

It seems fairly clear that the DQN AI is able to learn under these conditions. That said the AI team with purely individualistic incentives had the highest win rate followed by the team only rewarded when winning. The teams where an agent gained reward when an ally took an action did far worse.

I suspect that the reason teams 0 and 5 were able to develop a strategy that had a higher win rate was because of the clarity of the signal they got from their simpler utility reward. Team 0 got rewarded for obvious actions (goal/capture) that could be associated with a chain of past actions and situations; furthermore these actions both support (captures) and directly lead (goals) to winning. Team 5 had one clear utility signal to learn from (winning) this could then be clearly applied to the chain of actions/situations that lead to that result.

For agents gaining reward from team action, the reward's origin may not have been sufficiently clear to drive useful learning. It is possible a more sophisticated agent could mentally model their team's actions, and would then be less impeded by team rewards, and perhaps even improved. Future work may want to consider investigating this, exploring the complexity of the agents, training time and method of utility signal delivery. However this is only a hypothesis based on the data generated and our own understanding of DQN systems.

A sharper test to evaluate the hypothesis that mixed rewards impede learning would be to compare two agents, one of whom receives only rewards for their actions, and the other also receives random rewards. If this hypothesis is correct, receiving random rewards should impede performance, which would parallel some observations from cognitive psychology.

We also suggest for further work that teams of mixed agents might also be considered. Other things to consider could be the effects of rewards for actions that are actively harmful to success.

There is a certain desire for the team oriented method is the correct path[3]; however for this situation the data is clear: an individualistic reward for success oriented actions or a simple singular reward for a target condition work better.

[3] We would like to thank Dr. Glenn Chappell for this observation.

Acknowledgements

I would like to thank several people and organizations for their help:
Dr. Orion Lawlor, Dr. Jonathan Metzgar, and Dr. Glenn Chappell for their work as my graduate committee.

UAF in general and the UAF CS department in particular for teaching me, providing work and offering this opportunity in the first place.

Jonathan Halford IV for his help with charts and plotting.

Dr. Cheng-fu Chen for encouraging me to pursue a masters degree in the first place.

My family for their support and encouragement.

Github

Google Deepmind's OpenSpiel (Lanctot et al, 2019):
https://github.com/deepmind/open_spiel

Futlol game files and other project files:
https://github.com/Hoyr/futlol

References

Anthony, Thomas, et al. “Learning to Play No-Press Diplomacy with Best Response Policy Iteration.” *ArXiv:2006.04635 [Cs, Stat]*, Aug. 2020. *arXiv.org*, http://arxiv.org/abs/2006.04635.

Ivakhnenko, A. G., and V. G. Lapa. "Cybernetic Predictive Devices." CCM Information Corporation. (1965).

Jaderberg, Max, et al. “Human-Level Performance in 3D Multiplayer Games with Population-Based Reinforcement Learning.” *Science*, vol. 364, no. 6443, May 2019, pp. 859–65. *DOI.org (Crossref)*, doi:10.1126/science.aau6249.

Lanctot, Marc, et al. “OpenSpiel: A Framework for Reinforcement Learning in Games.” *ArXiv:1908.09453 [Cs]*, Sept. 2020. *arXiv.org*, http://arxiv.org/abs/1908.09453.

Leibo, Joel Z., et al. “Multi-Agent Reinforcement Learning in Sequential Social Dilemmas.” *ArXiv:1702.03037 [Cs]*, Feb. 2017. *arXiv.org*, http://arxiv.org/abs/1702.03037.

Minsky, Marvin, and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. 2. print. with corr, The MIT Press, 1972.

Munn, Luke. “Angry by Design: Toxic Communication and Technical Architectures.” *Humanities and Social Sciences Communications*, vol. 7, no. 1, Dec. 2020, p. 53. *DOI.org (Crossref)*, doi:10.1057/s41599-020-00550-7.

OpenAI, et al. “Dota 2 with Large Scale Deep Reinforcement Learning.” *ArXiv:1912.06680 [Cs, Stat]*, 1, Dec. 2019. *arXiv.org*, http://arxiv.org/abs/1912.06680.

Paquette, Philip, et al. “No Press Diplomacy: Modeling Multi-Agent Gameplay.” *ArXiv:1909.02128 [Cs]*, Nov. 2019. *arXiv.org*, http://arxiv.org/abs/1909.02128.

Rosenblatt, Frank. “The Perceptron—a perceiving and recognizing automaton”. *Report 85-460-1*. Cornell Aeronautical Laboratory. Jan, 1957.

Watkins, Christopher JCH, and Peter Dayan. "Q-learning." *Machine learning* 8.3-4 (1992): 279-292.